Ordered Binary Decision Diagrams Representing Knowledge-Bases (Models of Computation and Algorithms)

AUTHOR(S):

Horiyama, Takashi; Ibaraki, Toshihide

数理解析研究所講究録
1093巻 1999年 93-98

Ordered Binary Decision Diagrams Representing Knowledge-Bases

Takashi HORIYAMA and Toshihide IBARAKI
堀山 貴史 茨木 俊秀
Department of Applied Mathematics and Physics, Kyoto University

Abstract. We propose to make use of an ordered binary decision diagram (OBDD) as a means of realizing knowledge-bases. We show that the OBDD-based representation is more efficient and suitable in some cases, compared with traditional CNF-based and/or model-based representations in the sense of space requirement. We then consider recognition problems of OBDDs, and present polynomial time algorithms for testing whether a given OBDD represents a positive (i.e., monotone) Boolean function, and whether it represents a Horn function.

1 Introduction

Logical formulae are one of the traditional means of representing knowledge in AI [8]. However, it is known that deduction from a set of propositional clauses is co-NP-complete and abduction is NP-complete [9]. Recently, an alternative way of representing knowledge, i.e., by a subset of its models, which are called characteristic models, has been proposed [4, 5, 6]. Deduction from a knowledge-base in this model-based approach can be performed in linear time, and abduction is also performed in polynomial time [4].

In this paper, we propose yet another method of knowledge representation, i.e., the use of ordered binary decision diagrams (OBDDs) [1, 2]. An OBDD is a directed acyclic graph representing a Boolean function, which can be considered as a variant of decision tree. By restricting the order of variable appearance and sharing isomorphic subgraphs, OBDDs have the following useful properties: 1) When a variable ordering is given, OBDD has a reduced canonical form for each Boolean function. 2) Many practical Boolean functions can be compactly represented. 3) There are efficient algorithms for Boolean operations on OBDDs. As a result of these properties, OBDDs are widely used for various applications, especially in computer-aided design and verification of digital systems [3, 10]. The manipulation of knowledge-bases by OBDDs, e.g., deduction and abduction, was first discussed by Madre and Coudert [7].

We first compare the above three representations, i.e., formula-based, model-based, and OBDD-based, on the basis of their sizes. In particular, we show that, in some cases, OBDDs require exponentially smaller space than other two representations, while there are also cases in which each of the other two requires exponentially smaller space. In other words, these three representations are orthogonal with respect to space requirement.

It is known that OBDD is efficient for knowledge-base operations such as deduction and abduction [7]. We investigate fundamental recognition problems of OBDDs, that is, testing whether a given OBDD represents a positive Boolean function, and testing whether it represents a Horn function. We often encounter these recognition problems, since the knowledge-base representing some real phenomenon is sometimes required to be positive (or Horn), from the hypothesis posed on the phenomenon and/or from the investigation of the mechanism causing the phenomenon. We show that these recognition problems for OBDDs can be solved in polynomial time for both positive and Horn cases.

The rest of this paper is organized as follows. The next section gives fundamental definitions and concepts. We compare the three representations in Section 3, and consider the problems of recognizing positive and Horn OBDDs in Sections 4 and 5, respectively.

2 Preliminaries

2.1 Notations and Fundamental Concepts

We consider a Boolean function $f : \{0,1\}^n \to \{0,1\}$. An *assignment* is a vector $a \in \{0,1\}^n$, whose i-th coordinate is denoted by a_i. A *model of* f is a satisfying assignment of f, and the *theory* $\Sigma(f)$ *representing* f is the set of all models of f. Given $a, b \in \{0,1\}^n$, we denote by $a \leq b$ the usual bitwise ordering of assignments; i.e., $a_i \leq b_i$ for all $i = 1, 2, \ldots, n$, where $0 < 1$. Given a subset $E \subseteq \{1, 2, \ldots, n\}$, χ^E denotes the characteristic vector of E; i.e., the i-th coordinate ${\chi^E}_i$ equals 1 if $i \in E$ and 0 if $i \notin E$.

Let $x_1, x_2, \ldots, x_n$ be n variables of f. Negation of variable x_i is denoted by $\overline{x}_i$. Any Boolean function can always be represented by some CNF (conjunctive normal form), which may not be unique. We sometimes do not make a distinction among a function f, its theory $\Sigma(f)$, and a CNF φ that represents f, unless confusion arises. We define a *restriction* of f by replacing a variable x_i by a constant $a_i \in \{0,1\}$, and denote it by $f|_{x_i=a_i}$. Namely, $f|_{x_i=a_i}(x_1, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, a_i, x_{i+1}, \ldots, x_n)$. The restriction may be applied to many variables. We also define $f \leq g$ (resp., $f < g$) by $\Sigma(f) \subseteq \Sigma(g)$ (resp., $\Sigma(f) \subset \Sigma(g)$).

A Boolean function f is *positive* if $f(a) \leq f(b)$ holds for all assignments a and b such that $a \leq b$. A theory Σ is *positive* if Σ represents a positive function. A clause is *positive* if all literals in the clause are positive, and a CNF is *positive* if it contains only positive clauses. It is known that a theory Σ is positive if and only if Σ can be represented by some positive CNF.

A theory Σ is *Horn* if Σ is closed under operation $\wedge_{bit}$, where $a \wedge_{bit} b$ is bitwise AND of models a and b. For example, if $a = (0011)$ and $b = (0101)$, then $a \wedge_{bit} b = (0001)$. The closure of a theory Σ with respect to $\wedge_{bit}$ is denoted by $Cl_{\wedge_{bit}}(\Sigma)$. We also use the operation $\wedge_{bit}$ as a set operation; $\Sigma(f) \wedge_{bit} \Sigma(g) = \{a \mid a = b \wedge_{bit} c$ holds for some $b \in \Sigma(f)$ and $c \in \Sigma(g)\}$. We often denotes $\Sigma(f) \wedge_{bit} \Sigma(g)$ by $f \wedge_{bit} g$, for convenience. Note that two functions $f \wedge g$ and $f \wedge_{bit} g$ are different.

A Boolean function f is *Horn* if $\Sigma(f)$ is Horn; equivalently if $f \wedge_{bit} f = f$ holds. A clause is *Horn* if the number of positive literals in it is at most one, and a CNF is *Horn* if it contains only Horn clauses. It is known that a theory Σ is Horn if and only if Σ can be represented by some Horn CNF.

The set of *characteristic models* of a Horn theory Σ, denoted by $Char(\Sigma)$, is given by $Char(\Sigma) = \{a \in \Sigma \mid a \notin Cl_{\wedge_{bit}}(\Sigma - \{a\})\}$. $Char(\Sigma)$ is uniquely defined for every Horn theory Σ, and satisfies $Cl_{\wedge_{bit}}(Char(\Sigma)) = \Sigma$.

2.2 Ordered Binary Decision Diagrams

An *ordered binary decision diagram* (OBDD) is a directed acyclic graph that represents a Boolean function. It has two sink nodes 0 and 1, called the *0-node* and the *1-node* respectively (they are together called *constant nodes*). Other nodes v are called *variable nodes*, and are labeled by variables x_i. Let $var(v)$ denote the label of node v. Each variable node has exactly two outgoing edges, called a *0-edge* and a *1-edge* respectively. One of the variable nodes becomes the unique source node. Let $X = \{x_1, x_2, \ldots, x_n\}$ denote the set of n variables. A *variable ordering* is a total ordering $(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$, associated with each OBDD, where π is a bijection $\{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$. The *level*[1] of a node v with label $x_{\pi(i)}$, denoted by $level(v)$, is defined to be $n - i + 1$. By convention, the level of constant nodes is defined to be 0. On every path from the root node to a constant node in an OBDD, each variable appears at most once, and the appearing variables are arranged in the decreasing order of their levels.

Every node v of an OBDD also represents a Boolean function f_v, defined by the subgraph consisting of those edges and nodes reachable from v. If node v is a constant node, f_v equals its label. If node v is a variable node, f_v is defined as $\overline{var(v)}\, f_{0\text{-}succ(v)} \vee var(v)\, f_{1\text{-}succ(v)}$ by Shannon's expansion, where $0\text{-}succ(v)$ and $1\text{-}succ(v)$ respectively denote the node pointed by the 0-edge and the 1-edge of node v. The function f represented by an OBDD is the one represented by the root node. Given an assignment a, the value of $f(a)$ is determined by following the corresponding path from the root node to a constant node in the

[1] This definition of level may be different from its common use.

following manner: at a variable node v, one of the outgoing edges is selected according to the assignment $a_{var(v)}$ to the variable $var(v)$. The value of the function is the label of the final constant node.

When two nodes u and v in an OBDD represent the same function, and their levels are the same, they are called *equivalent*. A node whose 0-edge and 1-edge both point to the same node is called *redundant*. An OBDD which has no equivalent nodes and no redundant nodes is *reduced*. The *size* of an OBDD is the number of nodes in the OBDD. Given a function f and a variable ordering, its reduced OBDD is unique and has the minimum size among all OBDDs with the same variable ordering. The minimum sizes of OBDDs representing a given Boolean function depends on the variable orderings [2]. In the following, we assume that all OBDDs are reduced.

3 Three Approaches for Knowledge-Base Representation

In this section, we compare three knowledge-base representations: CNF-based, model-based, and OBDD-based. It is known that CNF-based and model-based representations play orthogonal roles with respect to space requirement. We show that OBDD-based representation is orthogonal to other two in the same sense. We start with relations between OBDD and CNF.

Lemma 3.1 *There exists a Horn theory on n variables, for which OBDD and CNF both require size $O(n)$, while its characteristic models require size $\Omega(2^{n/2})$.*

Lemma 3.2 *There exists a Horn theory on n variables, for which OBDD requires size $O(n)$ and characteristic models require size $O(n^2)$, while CNF requires size $\Omega(2^{n/2})$.*

Theorem 3.1 *There exists a Horn theory on n variables, for which OBDD requires size $O(n)$, while both of the characteristic models and CNF require sizes $\Omega(2^{n/4})$.*

We now turn to the opposite direction.

Theorem 3.2 *There exist a Horn theory on n variables, for which CNF requires size $O(n)$ and characteristic models require size $O(n)$, while the size of the smallest OBDD representation is $\Omega(2^{\sqrt{n}/\sqrt{2}})$.*

The above results show that none of the three representations always dominate the other two. OBDDs can find a place in knowledge-bases as they can represent some theories more efficiently than others.

4 Checking Positivity of OBDD

In this section, we discuss the problem of checking whether a given OBDD is positive. The following well-known property indicates that this problem can be solved in polynomial time.

Property 4.1 *Let f be a Boolean function on n variables $x_1, x_2, \ldots, x_n$. Then, f is positive if and only if $f|_{x_i=0} \leq f|_{x_i=1}$ holds for all $i = 1, 2, \ldots, n$.*

An OBDD representing $f|_{x_i=0}$ (resp., $f|_{x_i=1}$) can be obtained in $O(|f| \log |f|)$ time, where $|f|$ denotes the size of the OBDD representing f [2]. The size does not increase by a restriction. Since property $g \leq h$ can be checked in $O(|g| \cdot |h|)$ time, the positivity of f can be checked in $O(n|f|^2)$ time by checking $f|_{x_i=0} \leq f|_{x_i=1}$ for all $i = 1, 2, \ldots, n$.

The following well-known property is useful to reduce the computation time.

Property 4.2 *Let f be a Boolean function on n variables $x_1, x_2, \ldots, x_n$. Then, f is positive if and only if both $f|_{x_n=0}$ and $f|_{x_n=1}$ are positive and $f|_{x_n=0} \leq f|_{x_n=1}$.*

The positivity of functions $f|_{x_n=0}$ and $f|_{x_n=1}$ can be also checked by applying Property 4.2 recursively. Algorithm CHECK-POSITIVE in Fig. 1 determines the positivity of f in the bottom-up manner (i.e., from the smallest level ($\ell = 1$) to the level of the root node) by checking positivity of all nodes. Note that the property $f|_{x_n=0} \leq f|_{x_n=1}$ can be also checked in the bottom-up manner, since it holds if and only if $f|_{x_n=0, x_{n-1}=a_{n-1}} \leq f|_{x_n=1, x_{n-1}=a_{n-1}}$ holds for both $a_{n-1} = 0$ and 1.

In Step 2, the positivity of f_v is easily checked, since both $f_{0\text{-}succ(v)}$ (i.e., $f_v|_{x_\ell=0}$) and $f_{1\text{-}succ(v)}$ (i.e., $f_v|_{x_\ell=1}$) have already been checked to be positive, and $f_{0\text{-}succ(v)}$ and $f_{1\text{-}succ(v)}$ have been compared in Step

Algorithm CHECK-POSITIVE

Input: An OBDD representing f with a variable ordering $(x_n, x_{n-1}, \ldots, x_1)$.
Output: "yes" if f is positive; otherwise, "no".

Step 1. Set $\ell := 1$.

Step 2. For each node v labeled with x_ℓ, check whether the function f_v is positive, where f_v is the function represented by node v. Namely, check whether $f_{0\text{-}succ(v)}$ and $f_{1\text{-}succ(v)}$ are positive, and whether $f_{0\text{-}succ(v)} \leq f_{1\text{-}succ(v)}$ holds. If there exists at least one node which is not positive, output "no" and halt.

Step 3. For each pair of nodes u and v such that $level(u) \leq \ell$ and $level(v) \leq \ell$, and at least one of $level(u)$ and $level(v)$ is equal to ℓ, check whether $f_u \leq f_v$ holds by checking $f_u|_{x_\ell=0} \leq f_v|_{x_\ell=0}$ and $f_u|_{x_\ell=1} \leq f_v|_{x_\ell=1}$.

Step 4. If $\ell = n$, where n is the level of the root node, then output "yes" and halt. Otherwise set $\ell := \ell + 1$ and return to Step 2.

Figure 1: An algorithm to check positivity of an OBDD.

3 of the previous iteration. Note that both 0 and 1 are considered to be positive functions.

In Step 3, comparison between f_u and f_v is also performed easily, since the comparisons between $f_u|_{x_\ell=a_\ell}$ and $f_v|_{x_\ell=a_\ell}$ for both $a_\ell = 0$ and 1 have already been completed. Note that $f_v|_{x_\ell=0} = f_{0\text{-}succ(v)}$ and $f_v|_{x_\ell=1} = f_{1\text{-}succ(v)}$ hold if $level(v) = \ell$, and $f_v|_{x_\ell=0} = f_v|_{x_\ell=1} = f_v$ holds if $level(v) < \ell$. Also note that $f_u = f_v$ holds if and only if u and v are the same node. After Step 3 is done for some ℓ, we know the results of comparisons between f_u and f_v for all pairs of nodes u and v such that $level(u) \leq \ell$ and $level(v) \leq \ell$. We store all the results, although some of them may not be needed.

Next, we consider the computation time of this algorithm. In Step 2, checking positivity for each node is performed in a constant time from the data computed in Step 3. The positivity is checked for all nodes. In Step 3, the comparison between f_u and f_v for each pair of nodes u and v is also performed in a constant time. The number of pairs compared in Step 3 during the entire computation is $O\left(\binom{|f|}{2}\right) = O(|f|^2)$, and this requires $O(|f|^2)$ time.

Theorem 4.1 *Given an OBDD representing a Boolean function f, checking whether f is positive is performed in $O(|f|^2)$ time, where $|f|$ is the size of the given OBDD.*

5 Checking Horness of OBDD

5.1 Conditions for Horness

In this section, we discuss the problem of checking whether a given OBDD is Horn. We assume, without loss of generality, that the variable ordering is always $(x_n, x_{n-1}, \ldots, x_1)$. Denoting $f|_{x_n=0}$ and $f|_{x_n=1}$ by f_0 and f_1 for simplicity, f is given by $f = \overline{x}_n f_0 \vee x_n f_1$, where f_0 and f_1 are Boolean functions on $n-1$ variables $x_1, x_2, \ldots, x_{n-1}$. Similar to the case of checking positivity, we can check the Horness of f by scanning Horness of all nodes in the bottom-up manner.

Lemma 5.1 *Let f be a Boolean function on n variables $x_1, x_2, \ldots, x_n$, which are expanded as $f = \overline{x}_n f_0 \vee x_n f_1$. Then, f is Horn if and only if both f_0 and f_1 are Horn and $f_0 \wedge_{bit} f_1 \leq f_0$ holds.*

The Horness of f_0 and f_1 can be also checked by applying Lemma 5.1 recursively. The following lemma says that the condition $f_0 \wedge_{bit} f_1 \leq f_0$ can be also checked in the bottom-up manner.

Lemma 5.2 *Let f, g and h be Boolean functions on n variables, which are expanded as $f = \overline{x}_n f_0 \vee x_n f_1$, $g = \overline{x}_n g_0 \vee x_n g_1$ and $h = \overline{x}_n h_0 \vee x_n h_1$, respectively. Then, property $f \wedge_{bit} g \leq h$ holds if and only if $f_0 \wedge_{bit} g_0 \leq h_0$, $f_0 \wedge_{bit} g_1 \leq h_0$, $f_1 \wedge_{bit} g_0 \leq h_0$ and $f_1 \wedge_{bit} g_1 \leq h_1$ hold.*

Algorithm CHECK-HORN

Input: An OBDD representing f with a variable ordering $(x_n, x_{n-1}, \ldots, x_1)$.
Output: "yes" if f is Horn; otherwise, "no".

Step 1 (initialize). $\ell := 1$; $horn[v] := \begin{cases} \text{YES} & \text{if } v \in \{0,1\}; \\ * & \text{otherwise;} \end{cases}$

$bit\text{-}imp[u,v,w] := \begin{cases} \text{NO} & \text{if } (u,v,w) = (1,1,0); \\ \text{YES} & \text{if } u,v,w \in \{0,1\} \text{ and } (u,v,w) \neq (1,1,0); \\ * & \text{otherwise.} \end{cases}$

Step 2 (check Horness in level ℓ). For each node v labeled with x_ℓ, check whether the function f_v is Horn according to Fig. 3. Set the result YES or NO to $horn[v]$. If there exists at least one node which is not Horn, output "no" and halt.

Step 3 (compute *bit-imp* in level ℓ). For each triple (u,v,w) of nodes such that $level(u) \leq \ell$, $level(v) \leq \ell$ and $level(w) \leq \ell$, and at least one of $level(u)$, $level(v)$ and $level(w)$ is equal to ℓ, check whether $f_u \wedge_{bit} f_v \leq f_w$ holds according to Fig. 4. Set the result YES or NO to $bit\text{-}imp[u,v,w]$.

Step 4 (iterate). If $\ell = n$ then output "yes" and halt. Otherwise set $\ell := \ell + 1$ and return to Step 2.

Figure 2: An algorithm to check Horness of an OBDD.

YES	if all of $horn[0\text{-}succ(v)]$, $horn[1\text{-}succ(v)]$ and $bit\text{-}imp[0\text{-}succ(v), 1\text{-}succ(v), 0\text{-}succ(v)]$ are YES.
NO	otherwise.

Figure 3: Checking $horn[v]$ for a node v in Step 2.

YES	if all of $bit\text{-}imp[1\text{-}succ'(u), 1\text{-}succ'(v), 1\text{-}succ'(w)]$, $bit\text{-}imp[0\text{-}succ'(u), 0\text{-}succ'(v), 0\text{-}succ'(w)]$, $bit\text{-}imp[0\text{-}succ'(u), 1\text{-}succ'(v), 0\text{-}succ'(w)]$ and $bit\text{-}imp[1\text{-}succ'(u), 0\text{-}succ'(v), 0\text{-}succ'(w)]$ are YES.
NO	otherwise.

Figure 4: Checking $bit\text{-}imp[u,v,w]$ (i.e., $f_u \wedge_{bit} f_v \leq f_w$) for a triple of nodes (u,v,w) in Step 3.

5.2 Algorithm to Check Horness

Algorithm CHECK-HORN in Fig. 2 checks the Horness of a given OBDD in the bottom-up manner. We use an array $horn[v]$ to denote whether node v represents a Horn function or not, and a three-dimensional array $bit\text{-}imp[u,v,w]$ to denote whether $f_u \wedge_{bit} f_v \leq f_w$ holds or not; each element of the arrays stores YES, NO or $*$ (not checked yet). $horn[v] =$ YES (i.e., $\mathcal{M}_{level(v),*}(f_v) = f_v$) implies that f_v is Horn even if f_v is treated as a Boolean function on more than $level(v)$ variables. (Recall that OBDD is reduced; all the added variables are redundant.) Similarly, $bit\text{-}imp[u,v,w] =$ YES implies that $f_u \wedge_{bit} f_v \leq f_w$ holds even if f_u, f_v and f_w are treated as Boolean functions on ℓ ($\geq l_{max}$) variables, where l_{max} denotes the maximum level of the nodes u, v and w.

In Step 2 of Algorithm CHECK-HORN, $horn[v]$ can be computed in a constant time by Fig. 3, corresponding to Lemma 5.1. Note that all nodes v satisfy $f_v|_{x_{level(v)}=0} = f_{0\text{-}succ(v)}$ and $f_v|_{x_{level(v)}=1} = f_{1\text{-}succ(v)}$. Also note that $horn[0\text{-}succ(v)]$, $horn[1\text{-}succ(v)]$ and $bit\text{-}imp[0\text{-}succ(v), 1\text{-}succ(v), 0\text{-}succ(v)]$ have already been computed in the previous iteration.

Similarly, $bit\text{-}imp[u,v,w]$ in Step 3 can be computed in a constant time by Fig. 4, corresponding to Lemma 5.2. $0\text{-}succ'(v)$ (resp., $1\text{-}succ'(v)$) denotes $0\text{-}succ(v)$ (resp., $1\text{-}succ(v)$) if $level(v) = \ell$, but denotes

v itself if $level(v) < \ell$. This is because $f_v|_{x_\ell=0} = f_{0\text{-}succ(v)}$ and $f_v|_{x_\ell=1} = f_{1\text{-}succ(v)}$ hold if $level(v) = \ell$, and $f_v|_{x_\ell=0} = f_v|_{x_\ell=1} = f_v$ holds if $level(v) < \ell$. After Step 3 is done for some ℓ, we have the results for all triples (u, v, w) of nodes such that $level(u) \leq \ell$, $level(v) \leq \ell$ and $level(w) \leq \ell$, which contain all the information required in the next iteration, although some of them may not be needed.

Now, we consider the computation time of Algorithm CHECK-HORN. In Step 2, $horn[v]$ for each node v is computed in a constant time from the information obtained in the previous Step 3. The Horness is checked for all nodes. In Step 3, $bit\text{-}imp[u, v, w]$ for each triple of nodes (u, v, w) is also computed in a constant time. The number of triples to be checked in Step 3 during the entire computation is $O(|f|^3)$, where $|f|$ is the size of the given OBDD, and this requires $O(|f|^3)$ time. The time for the rest of computation is minor.

Theorem 5.1 *Given an OBDD representing a Boolean function f, checking whether f is Horn is performed in $O(|f|^3)$ time, where $|f|$ is the size of the given OBDD.*

6 Conclusion

In this paper, we considered to make use of OBDDs as knowledge-bases. We have shown that the three representations (i.e., CNF-based, model-based, and OBDD-based) play orthogonal roles with respect to space requirement. Thus, OBDDs can find a place in knowledge-bases as they can represent some theories more efficiently than others.

We then considered the problem of recognizing whether a given OBDD is positive, and whether it is Horn; checking positivity can be done in quadratic time of the size of OBDD, while checking Horness can be done in cubic time.

OBDDs are dominatingly used in the field of computer-aided design and verification of digital systems. The reason is that many Boolean functions which we encounter in practice can be compactly represented, and that many operations on OBDDs can be efficiently performed. We believe that OBDDs are also useful for manipulating knowledge-bases. Developing efficient algorithms for knowledge-base operations should be addressed in the further work.

References

[1] S.B. Akers, "Binary Decision Diagrams," *IEEE Trans. Comput.*, C-27, no.6, pp.509–516, 1978.

[2] R.E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. Comput.*, C-35, no.8, pp.677–691, 1986.

[3] J.R. Burch, E.M. Clarke, K.L. McMillan, and D.L. Dill, "Sequential Circuit Verification Using Symbolic Model Checking," *in Proc. of 27th ACM/IEEE DAC*, pp.46–51, 1990.

[4] H.A. Kautz, M.J. Kearns, and B. Selman, "Reasoning with Characteristic Models," *in Proc. of AAAI-93*, pp.34–39, 1993.

[5] H.A. Kautz, M.J. Kearns, and B. Selman, "Horn Approximations of Empirical Data," *Artificial Intelligence*, 74, pp.129–245, 1995.

[6] R. Khardon and D. Roth, "Reasoning with Models," *Artificial Intelligence*, 87, pp.187–213, 1996.

[7] J.C. Madre and O. Coudert, "A Logically Complete Reasoning Maintenance System Based on a Logical Constraint Solver," *in Proc. of IJCAI-91*, pp.294–299, 1991.

[8] J. McCarthy and P.J. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence," *in Machine Intelligence 4*, ed. D. Michie, Edinburgh University Press, 1969.

[9] B. Selman and H.J. Levesque, "Abductive and Default Reasoning: A Computational Core," *in Proc. of AAAI-90*, pp.343–348, 1990.

[10] N. Takahashi, N. Ishiura, and S. Yajima, "Fault Simulation for Multiple Faults Using BDD Representation of Fault Sets," *in Proc. of IEEE/ACM ICCAD-91*, pp.550–553, 1991.